Работа с пользовательскими функциями в PHP

Вы уже знаете многие стандартные функции PHP, например **substr(), reverse(), split()**. Их использование очень удобно - чтобы написать свои аналоги на PHP, потребовалось бы много времени и сил.

Однако, с помощью стандартных функций невозможно сделать все, что нам требуется. На помощь приходит такой механизм PHP, как **функции пользователя**. С их помощью мы можем создавать свои функции, принцип работы которых аналогичен стандартным функциям PHP.

Зачем нужны пользовательские функции?

Очень часто при программировании возникает такая ситуация: некоторые участки кода повторяются несколько раз в разных местах. Пользовательские Функции нужны для того, чтобы убрать дублирование кода.

Дублирование плохо тем, что если вам потребуется что-то поменять в коде - это придется сделать во многих местах При этом обязательно в каком-нибудь месте вы это сделать забудете. Практика копирования участков кода и вставки в другое место - очень **плохая практика** (очень часто ей грешат новички).

Кроме того, функции скрывают внутри себя какой-то код, о котором нам не приходится задумываться. Например, у нас есть функция **saveUser()**, которая сохраняет пользователя в базу данных. Мы можем просто вызывать ее, не задумываясь о том, какой код выполняется внутри функции. Это очень удобно. В программировании это называется *инкапсуляцией*.

Синтаксис функций пользователя

Функция создается с помощью команды **function**. Далее через пробел следует *имя функции* и *круглые скобки*. Круглые скобки могут быть **пустыми**, либо в них могут быть указаны **параметры**, которые принимает функция.Параметры - это обычные переменные PHP.

Сколько может быть параметров: один, несколько (в этом случае они указываются через запятую), ни одного (в этом случае круглые скобки все равно нужны, хоть они и будут пустыми).

```php
<?php

        //'func' - это имя функции, ей передается один параметр $param:

        function func($param)

        {


        }


        //Передаем функции два параметра - $param1 и $param2:

        function func($param1, $param2)
```

```
        {


        }


        //Вообще не передаем никаких параметров:
        function func()
        {


        }
?>
```

Обратите внимание на фигурные скобки в примерах: в них мы пишем **код**, который выполняет функция. Общий синтаксис функции выглядит так:

```
<?php
        function имя_функции(здесь перечисляются параметры через запятую)
        {
                Здесь написаны команды, которые выполняет функция.
        }
?>
```

Как вызвать функцию в коде?

Саму функцию мы можем вызвать в любом месте нашего PHP документа (*даже до ее определения*). Функция вызывается по ее **имени**. При этом круглые скобки после ее имени обязательны, даже если они пустые. Смотрите пример:

```
<?php
        /*
                Мы написали функцию,
                которая выводит на экран 'Привет, мир!'.
                Назовем ее 'hello':
        */
        function hello()
        {
                /*
                        В этом месте ничего не выведется само по себе,
                        а выведется только при вызове функции.
                */
                echo 'Привет, мир!';
        }


        /*
```

```
		Теперь мы хотим вызвать нашу функцию,
		чтобы она вывела на экран свою фразу.
		Обратимся к ней по имени:
	*/
	hello(); //Вот и все! Функция выведет на экран фразу 'Привет, мир!'.
?>
<?php
	/*
		Мы можем вывести нашу фразу несколько раз -
		для этого достаточно написать функцию
		не один раз, а несколько:
	*/
	hello();
	hello();
	hello();
?>
```

Подробнее о параметрах

В наших примерах мы вызывали функцию **'hello()'** без параметров.

Давайте теперь попробуем ввести параметр, который будет задавать текст, выводимый нашей функцией:

```
<?php
	//Зададим нашу функцию:
	function hello($text) { //укажем здесь параметр $text
		//Выведем на экран значение переменной $text:
		echo $text;
	}

	//Теперь вызовем нашу функцию:
	hello('Привет, Земляне!'); //она выведет именно ту фразу, которую мы ей передали
?>
```

Обратите внимание на переменную **$text** в нашей функции: в ней появляется то, что мы передали в круглых скобках при вызове функции.

Как PHP знает, что текст 'Привет, Земляне!' нужно положить в переменную **$text**? Мы сказали ему об этом, когда создавали нашу функцию, вот тут: **'function hello(text)'**.

Если мы зададим несколько параметров - то каждый из них будет лежать в своей переменной внутри функции.

Инструкция return

Чаще всего функция должна не выводить что-то через echo на экран, а *возвращать*.

Сделаем теперь так, чтобы функция не выводила что-то на экран, а **возвращала текст** скрипту, чтобы он смог записать этот текст в переменную и как-то обработать его дальше:

```
<?php
	//Зададим функцию:
	function hello($text)
	{
		/*
			Укажем функции с помощью инструкции 'return',
			что мы хотим, чтобы она ВЕРНУЛА текст, а не вывела на экран:
		*/
		return $text;
	}

	//Теперь вызовем нашу функцию и запишем значение в переменную message:
	$message = hello('Привет, Земляне!'); //пока вывода на экран нет

	//И теперь в переменной $message лежит 'Привет, Земляне!':
	echo $message; //убедимся в этом

	//Можно не вводить промежуточную переменную $message, а сделать так:
	echo hello('Привет, Земляне!');
?>
```

В принципе, практической пользы от того, что мы сделали - никакой, так как функция вернула нам то, что мы ей передали. Модернизируем наш пример так, чтобы функция принимала имя человека, а выводила строку 'Привет, %имя_человека_переданное_функции%!':

```
<?php
	//Зададим функцию:
	function hello($name) { //укажем здесь параметр $name, в котором будет лежать имя человека
		//Введем переменную $text, в которую запишем всю фразу:
		$text = 'Привет, '.name.'!';

		/*
			Укажем функции с помощью инструкции 'return',
			что мы хотим, чтобы она ВЕРНУЛА содержимое переменной $text:
```

```
		*/
		return $text;
	}

	//Теперь вызовем нашу функцию и запишем значение в переменную $message:
	$message = hello('Дима');

	//И теперь в переменной $text лежит 'Привет, Дима!':
	echo $message; //убедимся в этом

	//Поздороваемся сразу с группой людей из трех человек:
	echo hello('Вася'.' '.hello('Петя').' '.hello('Коля'));
?>
```

Частая ошибка с return

После того, как выполнится инструкция **return** – функция закончит свою работу. То есть: *после выполнения return больше никакой код не выполнится.*

Это не значит, что в функции должен быть **один return**. Но выполнится только один из них. Такое поведение функций является причиной многочисленных трудноуловимых ошибок.

Смотрите пример:

```
<?php
	function func($param)	{
		/*
			Если переменная $param имеет значение true, то вернем 'Верно!'.
			Напоминаю о том, что конструкция if($param) эквивалентна if($param === true)!
		*/
		if ($param) return 'Верно!';
		/*
			Далее новичок в PHP хочет проделать еще какие-то операции,
			но если переменная $param имеет значение true – сработает инструкция return,
			и код ниже работать не будет!

			Напротив, если $param === false – инструкция return не выполнится
			и код дальше будет работать!
```

```
                */
                echo 'Привет, мир!';
        }
        //Осознайте это и не совершайте ошибок
?>
```

Приемы работы с return

Существуют некоторые приемы работы с **return**, упрощающие код. Как сделать проще всего:

```
<?php
        function func($param)
        {
                /*
                        Что делает код:
                        если $param имеет значение true – то в переменную $result запишем 'Верно!',
                        иначе 'Неверно!':
                */
                if ($param) {
                        $result = 'Верно!'
                } else {
                        $result = 'Неверно!';
                }

                //Вернем содержимое переменной result:
                return $result;
        }
?>
```

Теперь упростим нашу функцию, используя прием работы с return:

```
<?php
        function func($param)
        {
                /*
                        Что делает код:
                        если $param имеет значение true – вернет 'Верно!',
                        иначе вернет 'Неверно!'.
                */
                if ($param) {
```

```
                return 'Верно!'
        } else {
                return = 'Неверно!';
        }
    }
?>
```

Обратите внимание на то, насколько упростился код. Плюсом также является то, что мы убрали лишнюю переменную **$result**.

Правильное использование пользовательских функций

Задачи на пользовательские функции

Примеры решения задач

Задача

Задача. Сделайте функцию, которая возвращает куб числа. Число передается параметром.

Решение:

```
<?php
    function cube($num)
    {
        return $num * $num * $num;
    }
?>
```

Задачи для решения

Простые функции

1. Сделайте функцию, которая возвращает квадрат числа. Число передается параметром.
2. Сделайте функцию, которая возвращает сумму двух чисел. Числа передаются параметрами функции.
3. Сделайте функцию, которая отнимает от первого числа второе и делит на третье.
4. Сделайте функцию, которая принимает параметром число от **1** до **7**, а возвращает день недели на русском языке.

Примеры решения задач

Задача

Задача. Дан массив с числами. Создайте из него новый массив, где останутся лежать только положительные числа. Создайте для этого вспомогательную функцию **isPositive**, которая параметром будет принимать число и возвращать true, если число положительное, и false - если отрицательное.

Решение:

```php
<?php
	$arr = [1, 2, 3, -1, -2, -3];

	function isPositive($num)
	{
		if ($num >= 0) {
			return true;
		} else {
			return false;
		}
	}

	$newArr = [];
	foreach ($arr as $elem) {
		if (isPositive($elem)) {
			$newArr[] = $elem;
```

```
            }
        }

        var_dump($newArr);
?>
```

Задачи для решения

1. Сделайте функцию **isNumberInRange**, которая параметром принимает число и проверяет, что оно больше нуля и меньше 10. Если это так - пусть функция возвращает **true**, если не так - **false**.

2. Дан массив с числами. Запишите в новый массив только те числа, которые больше нуля и меньше 10-ти. Для этого используйте вспомогательную функцию **isNumberInRange** из предыдущей задачи.

3. Сделайте функцию **getDigitsSum** (digit - это цифра), которая параметром принимает целое число и возвращает сумму его цифр.

4. Найдите все года от 1 до 2020, сумма цифр которых равна **13**. Для этого используйте вспомогательную функцию **getDigitsSum** из предыдущей задачи.

5. Сделайте функцию **isEven()** (even - это четный), которая параметром принимает целое число и проверяет: четное оно или нет. Если четное - пусть функция возвращает **true**, если нечетное - **false**.

6. Дан массив с целыми числами. Создайте из него новый массив, где останутся лежать только четные из этих чисел. Для этого используйте вспомогательную функцию **isEven** из предыдущей задачи.

7. Сделайте функцию **getDivisors**, которая параметром принимает число и возвращает массив его делителей (чисел, на которое делится данное число).

8. Сделайте функцию **getCommonDivisors**, которая параметром принимает 2 числа, а возвращает массив их общих делителей. Для этого используйте вспомогательную функцию **getDivisors** из предыдущей задачи.